Languages for Learning and Mining

Luc De Raedt
KU Leuven, Department of Computer Science
Celestijnenlaan 200A, POBox 2402
3001 Heverlee, Belgium

Abstract

Applying machine learning and data mining to novel applications is cumbersome. This observation is the prime motivation for the interest in languages for learning and mining. This note provides a gentle introduction to three types of languages that support machine learning and data mining: inductive query languages, which extend database query languages with primitives for mining and learning, modelling languages, which allow to declaratively specify and solve mining and learning problems, and programming languages, that support the learning of functions and subroutines. It uses an example of each type of language to introduce the underlying ideas and puts them into a common perspective. This then forms the basis for a short analysis of the state-of-the-art.

Introduction

Machine learning and data mining are popular subfields of artificial intelligence. However, it is well-known that applying machine learning and data mining to novel data sets is challenging because each application imposes its own requirements and constraints that often require the development of new algorithms and systems. While there are software packages and tools such as Scikit for machine learning and Weka, Orange or Knime for data mining, adapting them to novel tasks is not easy, which explains why one often resorts to implementing new algorithms and variations from scratch.

This observation is not new and it forms also the key motivation for different proposals for languages for learning and mining. For instance, (Imielinski and Mannila 1996) argue for the development of inductive query languages for data mining with *"a focus on increasing programmer productivity for KDD applications."* The ultimate goal of inductive databases and querying is to put data mining on the same methodological grounds as databases and to derive the equivalent of Codd's relational database model for data mining, that is, to find the primitives underlying data mining. (Imielinski and Mannila 1996) argue that patterns should be first class citizens that can be queried and manipulated just like data, and they used the slogan

> From the user point of view, there is no such thing as real discovery, just a matter of the expressive power of the available query language.

Coming more from a machine learning perspective, Tom Mitchell (2006) in his essay on *The Discipline of Machine Learning* asks a similar question but using programming languages rather than query languages and focussing on learning instead of mining:

> Can we design programming languages containing machine learning primitives? Can a new generation of computer programming languages directly support writing programs that learn? ... Why not design a new computer programming language that supports writing programs in which some subroutines are hand-coded while others are specified as to be learned? Such a programming language could allow the programmer to declare the inputs and outputs of each to be learned subroutine, then select a learning algorithm from the primitives provided by the programming language.

Finally, inspired by the field of constraint programming, (Guns et al. 2013) aim at developing declarative modeling languages for specifying a wide range of mining problems. Such languages should support

> the high-level and natural modeling of pattern mining tasks; that is, the models should closely correspond to the definitions of data mining problems found in the literature; should support user-defined constraints and criteria such that existing problem formulations can be extended and modified and novel mining tasks can be specified.

Although query, modelling and programming languages are quite different types of language, the motivation for incorporating machine learning and data mining primitives in these languages is essentially the same: it is to bring added power and expressiveness to the programmer who is developing machine learning and data mining software or applications. Furthermore, also the means with which they want to realize this goal are essentially the same: it is to provide the programmer with a formalism for declaratively specifying the patterns or models of interest. This should facilitate the task of the developer as providing a specification of the problem is much easier than implementing a full algorithm in some programming language.

Beer	Brand	Color	Alcohol%
1	Westmalle Tripel	Blonde	9.5
2	Orval	Dark	6.2
3	Straffe Hendrik	Gold	9
4	Straffe Hendrik	Dark	11
....	...	...	...

Table 1: The Beer table.

Cid	Brand	Color	Alcohol%
c1	Westmalle Tripel	?	?
c2	Westmalle Tripel	Blonde	?
c3	?	Blonde	?
c4	?	Blonde	9.5
c5	Straffe Hendrik	?	?
....	...	...	...

Table 2: The BeerConcept table.

This note will provide a gentle introduction to these three types of languages and will put them into a common perspective. This will allow us to clarify the state-of-the-art and to identify some remaining challenges. But first, we shall introduce these three types of languages using an example.

Inductive Query Languages

Since the seminal paper by (Imielinski and Mannila 1996) many different inductive query languages have been developed, e.g. (Meo, Psaila, and Ceri 1998; Imielinski and Virmani 1999; Blockeel et al. 2012). Most of these languages extend relational database languages (such as SQL) with primitives for mining. While some query languages merely invoke an extra primitive to call a particular data mining algorithm, such as a decision tree learner, others aim for a tighter integration between the database and data mining components. An interesting perspective in this regard is that of the virtual mining views of (Blockeel et al. 2012), who claim that inductive querying can be realized by adding a number of 'virtual views' to the database and querying these as any other relation in a database. A virtual mining view is a relation that virtually contains the output of a data mining component. It is virtual in that it need not be fully materialized as this would lead to combinatorial problems.

Let us illustrate this idea on a simple example (adapted from (Blockeel et al. 2012)). Assume we have a dataset about Belgian beers, cf. Table 1.

Two mining views could then be created for this relation, cf. Tables 2 and 3. The BeerConcept table lists all possible tuples using the values of the attributes that occur in the original table, but also allows for the special don't care value "?". The idea is that this virtual view contains the set of all possible patterns, where a pattern can be regarded as a conjunction of attribute-value pairs. The BeerSet table contains the corresponding patterns (through their concept identifier cid) as well as information about the number of occurences, i.e. the frequency, of these patterns in the original dataset. Once these tables are in place, one can mine for patterns using SQL on the resulting tables. For instance, the query below asks for all patterns (concepts) that occur at least 10 times in the data or have an area of at least 60 such as c5.

cid	frequency	size
c5	2	1
...	...	...

Table 3: The BeerSet table.

```
SELECT C.*, S.supp, S.sz,
S.supp * S.sz AS area
FROM BeerConcepts C, BeerSets S
WHERE (C.cid = S.cid AND (S.freq * S.sz > 60))
OR (S.freq > 10)
```

The concept of virtual mining views has been applied to richer types of patterns such as rules and decision trees, and it has been successfully integrated in a database system. Its integration in database systems is typical for inductive query languages. It involves processing the SQL query to split it in three parts corresponding to 1) pre-processing, 2) data mining, and 3) post-processing. The pre-processing query will retrieve the data from the database and put it in the right format so that it can be processed by a data mining system. The data mining system will then process the data and pass on the result to the post-processing step, which will query the data mining result to obtain the final answer.

Inductive querying is elegant and quite flexible, though many challenges remain. First, the integration of various types of patterns and models in a database management system remains cumbersome as it typically requires coupling with external data mining algorithms and there is only little support for this as deep integrations of the mining primitives in the underlying database management and query optimization system are still missing. Secondly, current inductive querying systems offer only little support to the user for defining new types of constraints and optimization criteria. Thirdly, despite the fact that inductive query languages are typically integrated in a (relational) database system, they often do not support the querying of relational or structured data. Finally, and most importantly, the 'equivalent' of the relational algebra for data mining has not yet been identified, and so the quest for inductive query languages continues.

Modeling Languages

In artificial intelligence there has been a shift from programming to solving as argued by (Geffner 2014). The idea is to model the artificial intelligence problem and then call a solver rather than to produce a program that implements a particular algorithm for computing the solutions to the problem. Numerous solvers have been developed for important classes of problems such as SAT, maxSAT, weighted model counting, Markov Decision Processes, constraint satisfaction problems and linear programming. Furthermore, these solvers enabled the development of high-level modeling languages in the knowledge representation and constraint programming communities such as Essence (Frisch et al. 2008) and Zinc (Marriott et al. 2008). It has been argued that this

type of approach represents (using the words of Eugene Freuder)

> ... one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.

and the slogan 'constraint programming = model + solver(s)' has been used.

Although solvers for convex optimization and mathematical programming are very popular in machine learning, the use of solvers for combinatorial optimization is less common in data mining and machine learning. More importantly, there are today only few high-level modeling approaches to machine learning or data mining (but see (Bruynooghe et al. 2014; Guns et al. 2013; De Raedt et al. 2011)) although the topic is enjoying more and more attention (cf. Dagstuhl Seminar 14411).

Let us now illustrate the idea underlying the modeling methodology. Figure 1 contains an example model written in MiningZinc (Guns et al. 2013), which is essentially a library that supports pattern mining for MiniZinc (Marriott et al. 2008), a popular modeling language for constraint programming. The model looks for discriminative patterns and it is specified in a form that closely resembles definitions of the discriminative pattern mining problem that one may find in the literature. It assumes that we are given a set of items $\{1, ..., N\}$, and two sets of transactions, `D_fraud` and `D_ok`, where each transaction (tid, I) consists of a transaction identifier tid and a subset I of the items $\{1, ..., N\}$. The goal then is to find an `Itemset` (i.e., a pattern) that 1) covers at least 5 transactions in `D_fraud` and 2) that scores best w.r.t. accuracy. An `Itemset` covers a transaction (tid, I) if `Itemset` $\subseteq I$ and the cover of an itemset is simply the set of transactions tid it covers. The constraint then uses the cardinality function `card` to represent the frequency constraint 1). The scoring function corresponds to the accuracy of the pattern (the difference between the frequencies in the two transaction sets) and the goal is to find the best pattern according to this scoring function.

The example illustrates the power of the modeling approach: by varying the constraints and the (scoring) functions in the model, one can easily specify a wide range of mining problems. To find solutions for any given model, one can either rely on general purpose solvers that are available for the underlying modeling language (MiniZinc), or automatically analyse the model in order to determine which data mining algorithm to use for solving particular subtasks of the problem following an approach that is analogous to that described for the virtual mining views. In the above example, one might, for instance, invoke a frequent pattern miner on `D_fraud` as one step in the solution process. MiningZinc provides support for incorporating such data mining solvers and for reasoning over possible executions of the solution methods. This is realized by defining each specific data mining solver as a conjunction of constraints.

Modeling languages allow to elegantly specify many problems. The modeled problems can – in theory – always be solved by resorting to one of the standard solvers, these solvers are usually not efficient enough and are certainly not

```
var set of 1..N: Itemset;
array[int] of set of 1..N: D_fraud;
array[int] of set of 1..N: D_ok;
constraint card(cover(Itemset,D_fraud)) > 5 ;
% Optimisation function
var int: Score = card(cover(Items, D_fraud)) -
                 card(cover(Items, D_ok));
solve maximize Score :: itemset_search(Items);
```

Figure 1: Discriminative itemset mining model

meant for dealing with the large datasets (and constraints) of data mining. The modeling languages can also be coupled to specific data mining algorithms but this requires extra effort. As for inductive query languages, there is not yet a deep integration in the underlying solvers (but see (Nijssen and Guns 2010) for a special purpose constraint programming solver for data mining). So, one of the most important challenges for the modeling approach to data mining is to devise solvers that are more efficient and scalable even though these more general approaches may never reach the performance of highy specialized solvers due to their greater expressivity. Another ongoing challenge is concerned with extending this type of modeling approach to a much wider spectrum of data mining tasks. A final one is, as for the inductive query languages, to support structured data such as sequences, graphs and relational data.

Probabilistic Programming Languages

Several extensions of programming languages exist that support learning from examples. Most popular amongst these are the so-called probabilistic programming languages (Sato and Kameya 2001; Goodman et al. 2008; Fierens et al. 2013; Milch et al. 2007; De Raedt and Kimmig 2013; McCallum, Schultz, and Singh 2009), which essentially define a probability distribution over possible executions and outcomes of the underlying program and then provide algorithms for estimating the parameters of the underlying distributions from samples of the desired behaviour of the program. This is very similar in spirit to the learning of graphical models from data, and indeed similar algorithms, such as EM and Bayesian methods, are employed. Most probabilistic programming languages extend existing functional or logic programming languages such as Scheme and Prolog with probabilistic primitives. Probabilistic programming languages have been applied to several challenging applications, and their key advantage is that it is easy to write down the model and then apply a learning engine. This is perfectly in line with the goal of supporting the programmer to write machine learning software.

Probabilistic programming has strong links to statistical relational learning (Getoor and Taskar 2007), though statistical relational learning approaches such as Markov Logic (Richardson and Domingos 2006) and Probabilistic Relational Models (Getoor et al. 2001) are not programming languages. While they possess many properties of the modeling approach described earlier they are also somewhat more limited in that they do not allow the user to model the learn-

```
0.1::burglary.
0.2::earthquake.
0.7::hears_alarm(mary).
0.4::hears_alarm(john).
alarm :- earthquake.
alarm :- burglary.
calls(x) :- alarm, hears_alarm(x).
call :- calls(x).
```

Figure 2: ProbLog program

ing task (e.g., the scoring function, the learning setting and learning algorithm are typically fixed and built-in).

Let us illustrate probabilistic programming using a ProbLog example (Fierens et al. 2013). ProbLog is a probabilistic extension of Prolog that allows for facts to be annotated with probability values. Each (ground) probabilistic fact $p :: f$ is viewed as an independent random variable f that is true with probability p and false with probability $1 - p$. Thus a ProbLog program defines a distribution over total choices, i.e., truth-assignments to the set of all (ground instances of the) probabilistic facts. In the example in Figure 2, these are the variables `burglary`, `earthquake`, `hears_alarm(mary)`, `hears_alarm(john)`. For instance, the probability that all of them except `earthquake` are true is $0.1 \times (1 - 0.2) \times 0.7 \times 0.4$. Each total choice then induces a (deterministic) Prolog program, and the set of all its consequences yields one possible world. The possible world corresponding to the above total choice includes also the facts `alarm`, `calls(john)`, `calls(mary)`, `call`. The probability of a query is defined as the probability that it is true in a randomly sampled possible world. For instance, the query `calls(mary)` has probability 0.196 as 0.196 is the sum of the probabilities of all the possibles worlds that contain `calls(mary)`.

The parameters of the probabilistic program could then be learned from samples of the target program, which could take the form of (partial) possible worlds or queries.

The key challenge for probabilistic programming is – not surprisingly – to perform efficient inference and learning and to scale up to large datasets. But there are also other open challenges. For instance, from a software engineering perspective one should be able to give guarantees on the performance of the learned programs. Will the answers they compute be approximately correct with high probability?

Further Languages for Learning

In addition to the three types of languages illustrated above, there exist several types of languages for mining and learning that have not been mentioned yet.

First, in addition to probabilistic programming languages, there exist also other types of programming languages that support learning. For instance, Learning Based Java (Rizzolo and Roth 2010) is an extension of Java that is very much in the spirit of Mitchell's vision as it allows to specify that certain procedures are to be learned from data. LBJ specifies and solves these problems using constraint conditional models. The way of realizing this in LBJ bears some resemblance to the modeling approach descibed in MiningZinc. It is only that the tasks are more machine learning oriented, related to predictive learning, and that the problems are cast as constraint conditional models instead of as constraint programs. Another example is Dyna (Eisner and Filardo 2011), which extends Datalog with semi-rings and learning, which makes it related to ProbLog though unlike ProbLog it associates weights to ground atoms rather than probabilities. This allows it to represent grammar-like structures but also neural networks, and it provides a gradient-based approach to parameter estimation. Finally, the kLog language and framework (Frasconi et al. 2014) integrates kernel-based learning with logic programs. It first turns the input data (represented as a logic program as in inductive logic programming) into a graph and then employs a graph-kernel on the resulting graphicalization to derive a set of features that are then used in an SVM. All of these languages have been used in natural language processing applications.

Another class of languages for learning has been developed with agent programming and reinforcement learning in mind. The language is used to specify the action and world model of the agent and reinforcement learning is used to determine the policies that the agent will execute. In this way, it combines elements of planning and Markov Decision Processes in a declarative modeling way. Example languages include ALISP (Andre and Russell 2002) and Golog (Beck and Lakemeyer 2012).

Discussion

This note has introduced three types of languages that support learning and mining. Although these types of languages are based on quite different principles (querying, modeling, and programming), these differences are not always as clear cut when looking at particular languages and there really seems to be a continuum. This is clear when looking, for instance, at some probabilistic programming languages, like ProbLog and Church (Goodman et al. 2008), whose programs could also be viewed as models.

What is more important than these differences is that they share a common goal: support the programmer to produce learning and mining software by writing queries, models or declarative programs and calling solvers rather than having to implement complex mining and learning algorithms. This is more convenient and more economical. For the machine learning and data mining scientists, these languages also realize rapid prototyping and in case the performance needs improvement, one can always implement a specialized algorithm implementing the model afterwards. Furthermore, developing machine learning and data mining systems within such a framework may also lead towards libraries of common problems and reuse of solutions.

Many of these languages have been used to solve real-life applications already. Furthermore, the several streams of research and the number of results in these area show that there is an increasing interest and an increasing need for such languages. However, they are also still limited in that they are typically based on a single paradigm or task, that they are not always tightly integrated into the inference engine of the underlying language, that they often need to be coupled to

an external data mining or learning system, and that it is not easy to scale up the inference and learning mechanisms to cope with realistic data sets. Thus the quest for the underlying primitives, the general principles and the universal solvers for learning and mining remains open.

Acknowledgments

The author is grateful to his collaborators, especially to Tias Guns, Siegfried Nijssen, and Angelika Kimmig, and also to Hendrik Blockeel and the reviewers for feedback on this note. This work was also supported by the EU ICON FP7 Project and by the BOF GOA project on Declarative Modeling for Learning and Mining.

References

Andre, D., and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In *Proceedings 18th National Conference on Artificial Intelligence*, 119–125.

Beck, D., and Lakemeyer, G. 2012. Reinforcement learning for golog programs with first-order state-abstraction. *Logic Journal of IGPL* 20(5):909–942.

Blockeel, H.; Calders, T.; Fromont, É.; Goethals, B.; Prado, A.; and Robardet, C. 2012. An inductive database system based on virtual mining views. *Data Mining and Knowledge Discovery* 24(1):247–287.

Bruynooghe, M.; Blockeel, H.; Bogaerts, B.; De Cat, B.; De Pooter, S.; Jansen, J.; Labarre, A.; Ramon, J.; Denecker, M.; and Verwer, S. 2014. Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with IDP3. *Theory and Practice of Logic Programming* 1–35.

De Raedt, L., and Kimmig, A. 2013. Probabilistic programming concepts. *arXiv preprint arXiv:1312.4328*.

De Raedt, L.; Nijssen, S.; O'Sullivan, B.; and Van Hentenryck, P. 2011. Constraint programming meets machine learning and data mining (dagstuhl seminar 11201). *Dagstuhl Reports* 1(5).

Eisner, J., and Filardo, N. W. 2011. Dyna: Extending datalog for modern ai. In *Datalog Reloaded*. Springer. 181–220.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2013. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming* 1–44.

Frasconi, P.; Costa, F.; De Raedt, L.; and De Grave, K. 2014. klog: A language for logical and relational learning with kernels. *Artificial Intelligence* in press.

Frisch, A. M.; Harvey, W.; Jefferson, C.; Martínez-Hernández, B.; and Miguel, I. 2008. Essence: A constraint language for specifying combinatorial problems. *Constraints* 13(3):268–306.

Geffner, H. 2014. Artificial intelligence: From programs to solvers. *AI Communications* 27(1):45–51.

Getoor, L., and Taskar, B., eds. 2007. *An Introduction to Statistical Relational Learning*. MIT Press.

Getoor, L.; Friedman, N.; Koller, D.; and Pfeffer, A. 2001. Learning probabilistic relational models. In Džeroski, S., and Lavrač, N., eds., *Relational Data Mining*. Springer. 307–335.

Goodman, N.; Mansinghka, V. K.; Roy, D. M.; Bonawitz, K.; and Tenenbaum, J. B. 2008. Church: a language for generative models. In *Proceedings 24th Conference on Uncertainty in Artificial Intelligence*, 220–229.

Guns, T.; Dries, A.; Tack, G.; Nijssen, S.; and De Raedt, L. 2013. Miningzinc: A modeling language for constraint-based mining. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 1365–1372. AAAI Press.

Imielinski, T., and Mannila, H. 1996. A database perspectivce on knowledge discovery. *Communications of the ACM* 39(11):58–64.

Imielinski, T., and Virmani, A. 1999. Msql: A query language for database mining. *Data Mining and Knowledge Discovery* 3(4):373–408.

Marriott, K.; Nethercote, N.; Rafeh, R.; Stuckey, P. J.; De La Banda, M. G.; and Wallace, M. 2008. The design of the zinc modelling language. *Constraints* 13(3):229–267.

McCallum, A.; Schultz, K.; and Singh, S. 2009. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, 1249–1257.

Meo, R.; Psaila, G.; and Ceri, S. 1998. An extension to sql for mining association rules. *Data Mining and Knowledge Discovery* 2(2):195–224.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2007. Blog: Probabilistic models with unknown objects. In Getoor, L., and Taskar, B., eds., *Statistical Relational Learning*. MIT Press.

Mitchell, T. M. 2006. *The discipline of machine learning*. Carnegie Mellon University, School of Computer Science, Machine Learning Department.

Nijssen, S., and Guns, T. 2010. Integrating constraint programming and itemset mining. In *Machine Learning and Knowledge Discovery in Databases*. Springer. 467–482.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62:107–136.

Rizzolo, N., and Roth, D. 2010. Learning based java for rapid development of nlp systems. In *Proceedings International Conference on Language Resources and Evaluation*.

Sato, T., and Kameya, Y. 2001. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* 15:391–454.